ezflags

Release 1.4.2

Feb 02, 2020

Contents:

CHAPTER 1

Installation

To install, you can use PyPI via pip, or you can also install directly from the GitHub repository. To install from PyPI:

```
pip install ezflags
```

To install from GitHub (This version might be unstable):

```
pip install git+https://github.com/karx1/ezflags.git@master
```

CHAPTER 2

Parsing flags

parse_flags takes in a list of strings denoting which flags to parse. If no list is given, it uses `sys.argv[1]`. If providing a list, the flags must be typed exactly as you would type them in the command line. For example:

```
import ezflags

parser = ezflags.FlagParser()
parser.add_flag(--flag, value=True)

flags = parser.parse_flags(["--flag"])
print(flags.flag) # Prints True
```

If your flag has a dash ("-") in it (excluding the dash(es) at the beginning), it is represented in python with an underscore. For example:

```
import ezflags
parser = ezflags.FlagParser()
parser.add_flag('--demo-flag', value=True, help="A demo flag.")

flags = parser.parse_flags()
print(flags.demo_flag) # Represents --demo-flag
```

CHAPTER 3

ezflags package

3.1 ezflags.flagparser module

class ezflags.flagparser.**FlagParser**(*program_name: str = None*, *description: str = None*, *epilogue: str = None*, *prefix_chars: str = None*, *debug: bool = False*, *debug_file=None*)

Bases: object

This is the main class for parsing flags.

Parameters

- **program_name** (*str, optional*) – The name of the program. Defaults to sys.argv[0]
- **description** (*str, optional*) – The message to display before the arguments.
- **epilogue** (*str, optional*) – The message to display at the end of the help message
- **prefix_chars** (*str, optional*) – The prefix of each argument. Defaults to '-'
- **debug** (*bool, optional*) – Turns on or off debug mode. Defaults to false.
- **debug_file** (*file, optional*) – The file to write to in debug mode. Needs to be a file object as returned by open. Defaults to :class'sys.stdout'.

flags A dictionary of flags and their values. For example:

```
{"--flag, -f": True}
```

add_flag(**args*, *value: bool*, *help: str = None*)

Add a flag to the parser.

Parameters

- **args** (*str*) – Things to name the flag. Maximum of two values.
- **value** (*bool*) – The value of the flag when present.

- **help** (*str, optional*) – A brief description of the flag. These descriptions will be displayed when the *-h* or *–help* flags are present.

parse_flags (*flag_list: List[str] = None*)
Parse the flag inputs. Returns an object with the values of each flag. See *Parsing flags* for more info.

Parameters **flag_list** (*list, optional*) – List of flags to parse. This can be used for testing. Defaults to `sys.argv[1:]`.

Returns Returns an object containing the values of all the flags.

Return type `_ParsedObj`

class ezflags.ext.exflagparser.**FlagParserExtended**(*program_name: str = None, description: str = None, epilogue: str = None, prefix_chars: str = None, debug: bool = False, debug_file=None*)

Bases: `argparse.ArgumentParser`

This is the class for using flags and argparse arguments in conjunction. It uses the same parameters as `FlagParser`.

Parameters

- **program_name** (*str, optional*) – The name of the program. Defaults to `sys.argv[0]`
- **description** (*str, optional*) – The message to display before the arguments.
- **epilogue** (*str, optional*) – The message to display at the end of the help message
- **prefix_chars** (*str, optional*) – The prefix of each argument. Defaults to '-'
- **debug** (*bool, optional*) – Turns on or off debug mode. Defaults to false.
- **debug_file** (*file, optional*) – The file to write to in debug mode. Needs to be a file object as returned by `open`. Defaults to :class'sys.stdout'.

flags A dictionary of flags and their values. For example:

```
{"--flag, -f": True}
```

add_flag (**args, value: bool, help: str = None, required: bool = False*)
Add a flag to the parser.

Parameters

- **args** (*str*) – Things to name the flag. Maximum of two values.
- **value** (*bool*) – The value of the flag when present.
- **required** (*bool, optional*) – Whether the flag is required for the script to run. Defaults to False.
- **help** (*str, optional*) – A brief description of the flag. These descriptions will be displayed when the *-h* or *–help* flags are present.

parse_flags (*flag_list: List[str] = None*) → argparse.Namespace
Parse the flag inputs. Returns an `argparse.Namespace` object with each flag. See *Parsing flags* for more info.

Parameters **flag_list** (*list, optional*) – List of flags to parse. This can be used for testing. Defaults to `sys.argv[1:]`.

Returns Returns an object containing the values of all the flags.

Return type Instance of `argparse.Namespace`

CHAPTER 4

Changelog

This is a detailed rendering of what changed in each version.

4.1 v1.4.2

- Create custom exceptions. This makes it easier to include ezflags in error handlers.
- Add str() and repr() functionality to _ParsedObj. This makes it easier to see what flags are present and their values.

4.2 v1.4.1

- Move FlagParserExtended to a new location, *ezflags.ext*
- Bring back logging

4.3 v1.4.0

- This update is a complete rewrite of the module.
- It moves away from using argparse and instead parses flags "in-house".
- The new class saves a lot of memory because it now only contains the logic for boolean flags.
- The argparse-like parser is still available through the FlagParserExtended class. (This may be moved to another location, so the import will be different.)
- The new parser class is still incomplete and may be buggy. Features will be brought back in the next series of incremental updates.

4.4 v1.3.3

- Add debug mode. This prints various messages about what the parser is currently doing. You can specify the file it writes to like this:

```
file = open("somefile.txt", "w")
parser = ezflags.FlagParser(debug=True, debug_file=file)
```

4.5 v1.3.2

- Change *parser.flags* to a dictionary with the flag names (and short versions, if applicable) corresponding to their value.
- Fix bug where an error was raised when providing only one flag name

4.6 v1.3.1

- Add *parser.flags_short*, which contains the short versions of each flag. Each index in *flags_short* corresponds to its longer counterpart in *parser.flags*.

4.7 v1.3.0

- You can now see a list of all the flags in the parser using *parser.flags*
- Performance improvements

4.8 v1.2.1

- Add a kwarg to specify whether the flag is required or not
- Limit to two flag names
- Improved type checking
- Take info from README and add it to index of documentation

4.9 v1.2.0

- Rename kwarg action to value
- Allow use of lists of flags in parse_flags(). Defaults to sys.argv[1:]

4.10 v1.1.4

- Create documentation

4.11 v1.1.3

- Add advanced functionality

4.12 v1.1.1

- Initial release

ezflags is a tool that makes creating command line flags super easy.

Similar to argparse, switching is no problem at all! You can even use the FlagParserExtended class as if it were a normal ArgumentParser for full integration with existing arguments.

Here's a simple example:

```
# main.py
import ezflags

parser = ezflags.FlagParser()
parser.add_flag('--flag', '-f', value=True, help="A demo flag.")

flags = parser.parse_flags()
print(flags.flag)
```

To integrate with ArgumentParser:

```
from ezflags.ext import FlagParserExtended

parser = FlagParserExtended()
parser.add_flag('--flag', '-f', value=True, help="A demo flag.")
parser.add_argument('--arg', '-a', help="A demo argument.")

args = parser.parse_args() # Flags are included, too!
print(args.flag)
print(args.arg)
```

This can be invoked as such:

```
python main.py --flag
# With ArgumentParser()
python main.py --flag --arg arg
```

Python Module Index

e

Index